GNU Assembler

CMPE230 - Spring'24

Gökçe Uludoğan

GNU Assembler

- the assembler developed by the GNU Project.
- used to assemble the GNU operating system and the Linux kernel.
- uses AT&T assembly syntax.

AT&T Syntax

- Similar to any other assembler syntax.
- Consists of a series of directives, labels, instructions
- Composed of a mnemonic followed by a maximum of three operands
  - **the ordering of the operands are reversed.**

Intel Syntax (e.g. A86)	mnemonic	destination, source
AT&T Syntax	mnemonic	source, destination

https://csiflabs.cs.ucdavis.edu/~ssdavis/50/att-syntax.htm

Registers

Register Names

64-bit register	32-bit sub-register	16-bit sub-register	8-bit sub-register
%rax	%eax	%ax	%al
%rbx	%ebx	%bx	%bl
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rsi	%esi	%si	%sil
%rdi	%edi	%di	%dil
%rbp	%ebp	%bp	%bpl
%rsp	%esp	%sp	%spl
%r8	%r8d	%r8w	%r8b
%r9	%r9d	%r9w	%r9b
%r10	%r10d	%r10w	%r10b
%r11	%r11d	%r11w	%r11b
%r12	%r12d	%r12w	%r12b
%r13	%r13d	%r13w	%r13b
%r14	%r14d	%r14w	%r14b
%r15	%r15d	%r15w	%r15b

https://web.stanford.edu/class/cs107/resources/x86-64-reference.pdf

Registers

Registers

`%rip`	Instruction pointer
`%rsp`	Stack pointer
`%rax`	Return value
`%rdi`	1st argument
`%rsi`	2nd argument
`%rdx`	3rd argument
`%rcx`	4th argument
`%r8`	5th argument
`%r9`	6th argument
`%r10,%r11`	Callee-owned
`%rbx,%rbp,` `%r12-%15`	Caller-owned

Instruction suffixes

`b`	byte
`w`	word (2 bytes)
`l`	long /doubleword (4 bytes)
`q`	quadword (8 bytes)

Suffix is elided when can be inferred from operands. e.g. operand `%rax` implies `q`, `%eax` implies `l`, and so on

https://web.stanford.edu/class/cs107/resources/x86-64-reference.pdf

Mnemonic	Purpose	Examples

mov *src,dest*	Move data between registers, load immediate data into registers, move data between registers and memory.	mov $4,%eax # Load constant into eax
		mov %eax,%ebx # Copy eax into ebx
		mov %ebx,123 # Copy ebx to mem. 123

push *src*	Insert a value onto the stack. Useful for passing arguments, saving registers, etc.	push %ebp
pop *dest*	Remove topmost value from the stack. Equivalent to "mov (%esp),*dest*; add $4,%esp"	pop %ebp
call *func*	Push the address of the next instruction and start executing func.	call print_int
ret	Pop the return program counter, and jump there. Ends a subroutine.	ret
add *src,dest*	*dest=dest+src*	add %ebx, %eax # Add ebx to eax
mul *src*	Multiply eax and *src*	mul %ebx #Multiply eax by ebx
jmp *label* jl *label*	Goto the instruction *label*:. Skips anything else in the way. Goto *label* if the comparison came out as less-than. Others: jle (<=), je (==), jge (>=), jg (>), jne (!=), and so on.	jmp post_mem mov %eax,0 # Write to NULL! post_mem: # OK here...
cmp *a,b*	Compare two values. Sets flags that are used by the conditional jumps (below).	cmp $10,%eax

GDB

- The GNU Debugger
- Install via

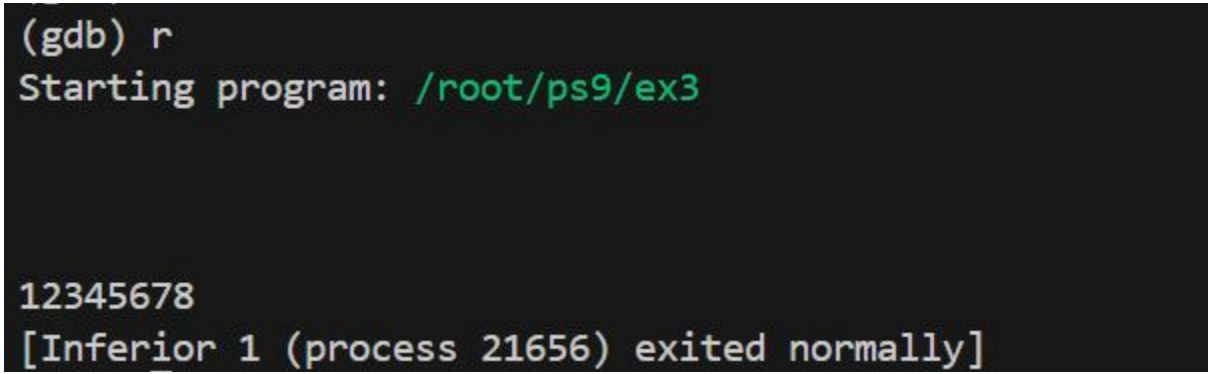
```
apt-get update
apt-get install gdb
```

- Run with

```
gcc <file>.s -c -g
ld <file>.o -o <program>
gdb <program>
```

GDB

`r` for running the program



GDB

`b` for setting a breakpoint

`s` for step

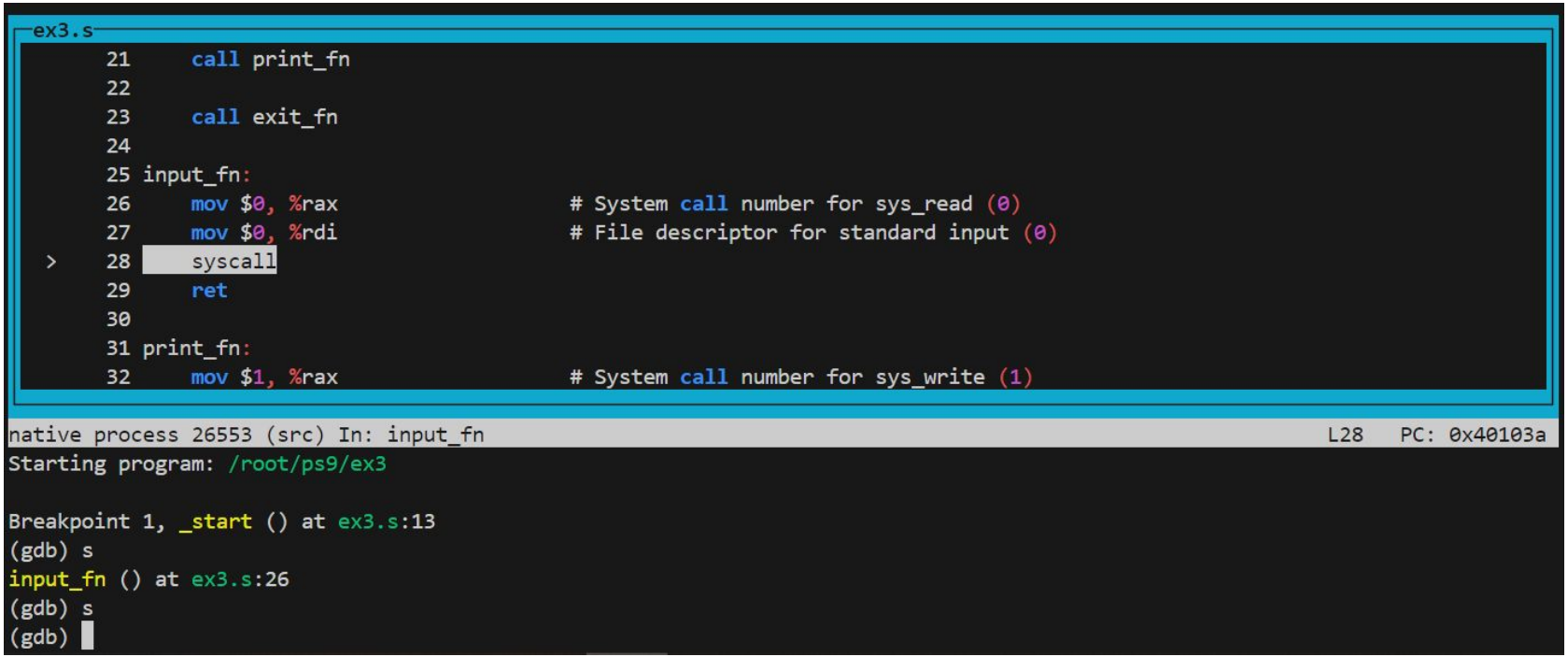
```
Reading symbols from ex3...
(gdb) br 1
Breakpoint 1 at 0x401000: file ex3.s, line 13.
(gdb) r
Starting program: /root/ps9/ex3

Breakpoint 1, _start () at ex3.s:13
13          lea input_buffer(%rip), %rsi
(gdb) s
14          mov $256, %edx
(gdb) s
15          call input_fn
```

GDB TUI

https://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_19.html

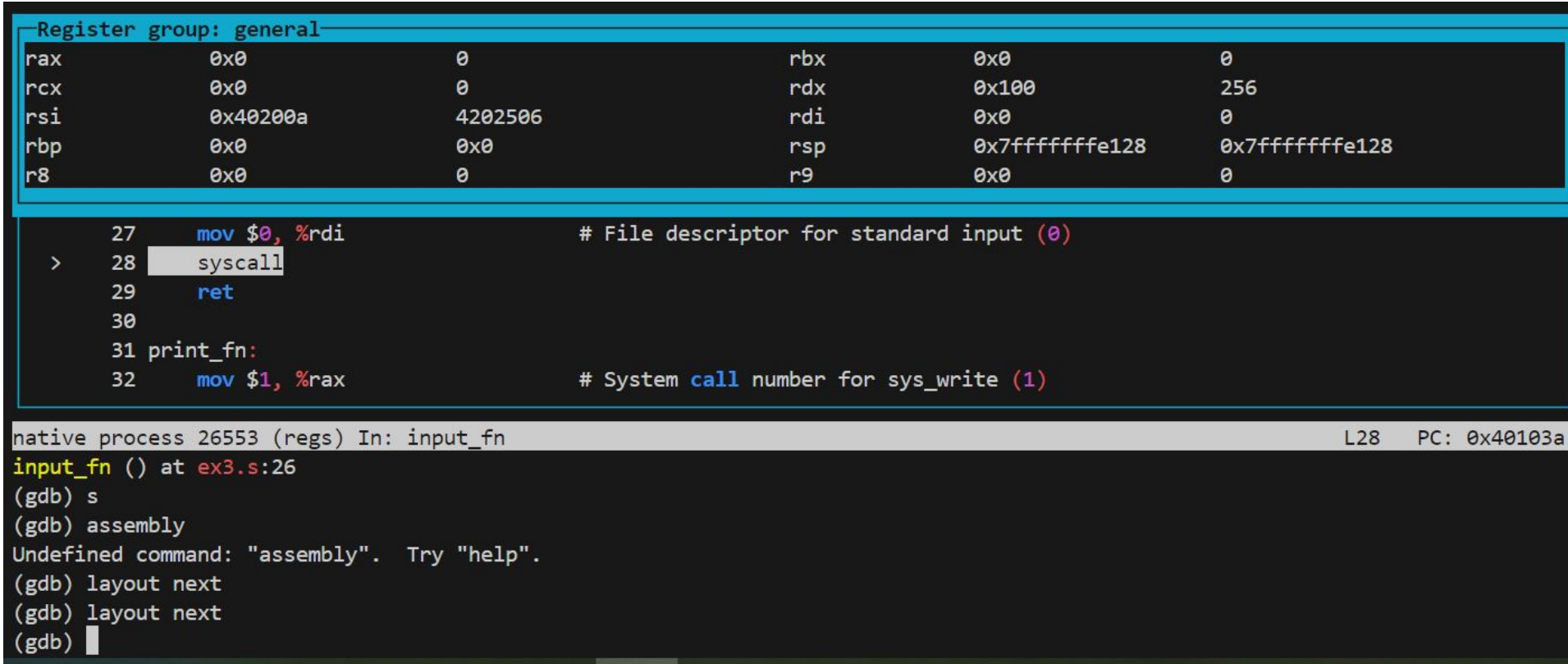
> gdb -tui <program>



GDB TUI Layouts

tui layout regs
tui layout asm

https://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_19.html



Example

- System calls

https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

```
.section .data
hello: .string "12345678\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer


.section .text
.global _start


_start:

    mov $0, %rax                    # System call number for sys_read (0)
    mov $0, %rdi                    # File descriptor for standard input (0)
    lea input_buffer(%rip), %rsi
    mov $256, %edx
    syscall

    mov $1, %rax                    # System call number for sys_write (1)
    mov $1, %rdi                    # File descriptor for standard output (1)
    syscall

    lea hello(%rip), %rsi
    mov $9, %edx

    mov $1, %rax                    # System call number for sys_write (1)
    mov $1, %rdi                    # File descriptor for standard output (1)
    syscall

    mov $60, %rax
    mov $0, %rdi
    syscall
```

Example

```
.section .data
hello: .string "12345678\n"

.section .bss
input_buffer: .space 256     # Reserve 256 bytes for input buffer


.section .text
.global _start


_start:
    lea input_buffer(%rip), %rsi
    mov $256, %edx
    call input_fn

    call print_fn

    lea hello(%rip), %rsi
    mov $9, %edx
    call print_fn

    call exit_fn

input_fn:
    mov $0, %rax                  # System call number for sys_read (0)
    mov $0, %rdi                  # File descriptor for standard input (0)
    syscall
    ret

print_fn:
    mov $1, %rax                  # System call number for sys_write (1)
    mov $1, %rdi                  # File descriptor for standard output (1)
    syscall
    ret

exit_fn:
    mov $60, %rax
    mov $0, %rdi
    syscall
```

Reverse string

```
.section .bss
input_buffer: .space 10            # Allocate 256 bytes for input buffer
output_buffer: .space 10

.section .text
.global _start

_start:
    # Read input from standard input
    mov $0, %eax                    # syscall number for sys_read
    mov $0, %edi                    # file descriptor 0 (stdin)
    lea input_buffer(%rip), %rsi    # pointer to the input buffer
    mov $10, %edx                  # maximum number of bytes to read
    syscall                         # perform the syscall

    lea input_buffer(%rip), %r15
    lea output_buffer(%rip), %r14
    add $2, %r14
    mov $0, %r13

reverse:
    mov $0, %rbx
    movb (%r15), %bl
    cmp $'\n', %rbx                 # Check if we've parsed all input
    je print_output_buffer          # If zero, finish the loop
    inc %r13

    mov %bl, (%r14)
    inc %r15
    dec %r14
    jmp reverse
```

```
print_output_buffer:
    lea output_buffer(%rip), %rsi
    mov %r13, %rdx
    mov $1, %eax                # syscall number for sys_write
    mov $1, %edi                # file descriptor 1 (stdout)
    syscall


exit_program:
    # Exit the program
    mov $60, %eax               # syscall number for sys_exit
    xor %edi, %edi              # exit code 0
    syscall
```

Reverse string using stack

```
.section .bss
input_buffer: .space 10            # Allocate 256 bytes for input buffer
output_buffer: .space 10

.section .text
.global _start

_start:

    # Read input from standard input
    mov $0, %eax                    # syscall number for sys_read
    mov $0, %edi                    # file descriptor 0 (stdin)
    lea input_buffer(%rip), %rsi    # pointer to the input buffer
    mov $10, %edx                  # maximum number of bytes to read
    syscall                         # perform the syscall

    lea input_buffer(%rip), %r15
    lea output_buffer(%rip), %r14
    mov $0, %r13

reverse:
    mov $0, %rbx
    movb (%r15), %bl
    cmp $'\n', %rbx                 # Check if we've parsed all input
    je store                        # If zero, finish the loop
    push %rbx
    inc %r13
    inc %r15
    jmp reverse
```

```
store:
    cmp $0, %r13
    je print_output_buffer
    dec %r13
    pop %rbx
    movb %bl, (%r14)
    inc %r14
    jmp store

print_output_buffer:
    lea output_buffer(%rip), %rsi
    mov %r14, %rdx
    mov $1, %eax                # syscall number for sys_write
    mov $1, %edi                # file descriptor 1 (stdout)
    syscall


exit_program:
    # Exit the program
    mov $60, %eax               # syscall number for sys_exit
    xor %edi, %edi              # exit code 0
    syscall
```

Output?

```
.section .data

n: .long 50

.section .text
.global _start


_start:
    mov $n, %rsi
    mov $1, %rdx

    call print_fn
    call exit_fn


print_fn:
    mov $1, %rax                # System call number for sys_write (1)
    mov $1, %rdi                # File descriptor for standard output (1)
    syscall
    ret

exit_fn:
    mov $60, %rax
    mov $0, %rdi
    syscall
```